Custom VBA Class Development

Class Modules

Ken Getz
keng@mcwtech.com
@kengetz

pluralsight
hardcore dev and IT training

Assumptions About You

- **You know how to write code in VBA**
- **You have written applications either in**
  - Visual Basic 4.0 or later, or
  - Microsoft Office 2002 or later (Word, Excel, or PowerPoint)
- **You need to create custom applications using VBA**
  - Content applies to Visual Basic.NET as well
- **You have used objects as part of VBA applications**
  - Debug, Screen, Application, Document, Workbook, and so on
- **Comfortable with concepts like properties and methods**
- **Can create and use object variables**

Why Use Class Modules?

- **Been using Visual Basic/VBA for a while?**
  - Might wonder "Why use class modules, anyway?"
- **Benefits vs. costs**
- **Primary cost:**
  - Learning curve required to create and use effectively
- **Benefits:**
  - Make your code more manageable, self-documenting, easier to maintain
  - Especially if working with complex sets of related data

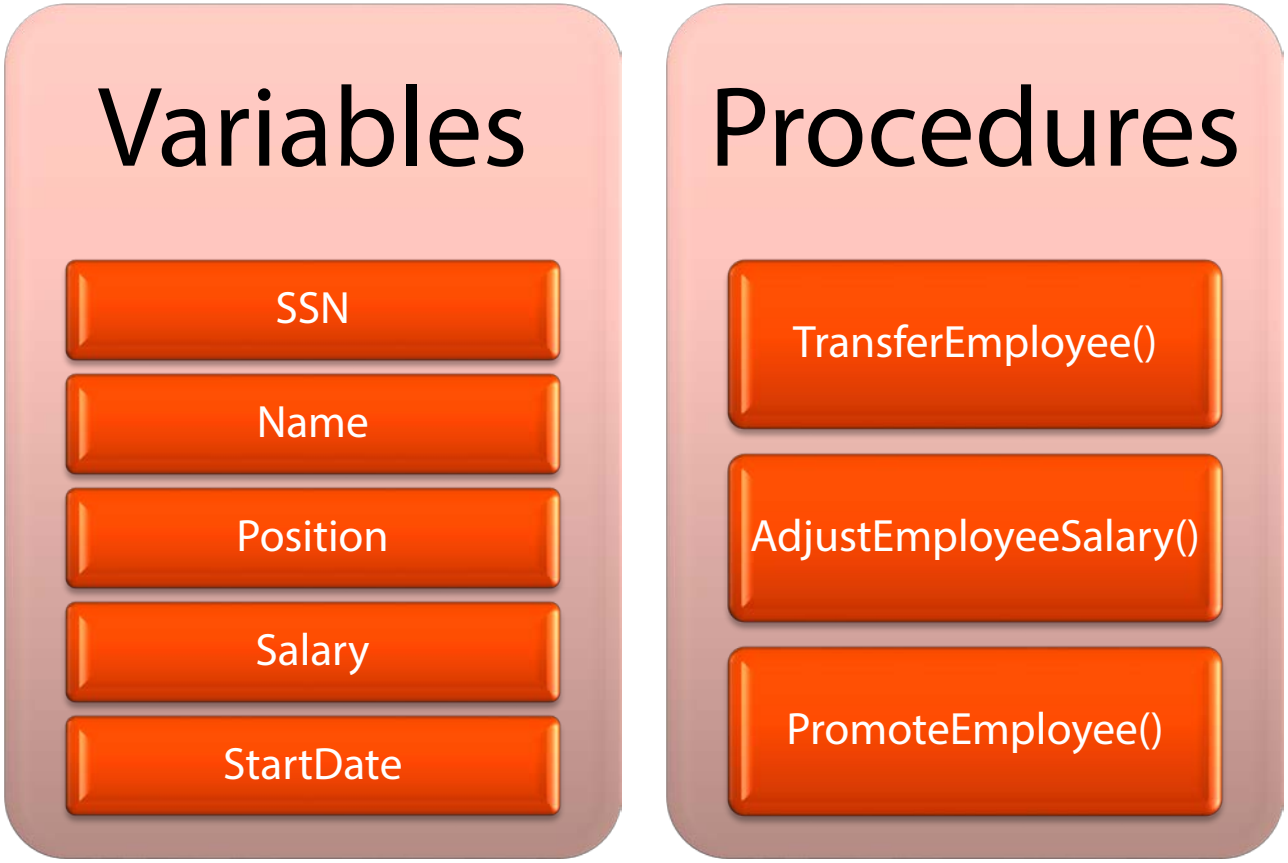
Encapsulate Data and Behavior

- **Main benefit of object-oriented programming and VBA classes**
  - Ability to encapsulate data and behavior in high-level constructs
- **Associate variables and procedures linked to a "thing"**
  - Make it a programmable entity
  - Entity is easily manipulated using VBA
  - Remains a discrete part of the application, never mingling with other entities
- **In essence, class modules allow you to create your own types**

Why Should You Care?

- Creating an application: tracks information about employees
- Using standard Basic, might create separate variables to store each employee's name, manager, salary
- Might create an array of user-defined data types
- Might create procedures to handle tasks like hiring and firing
- What ties all these bits of information together?

Without Class

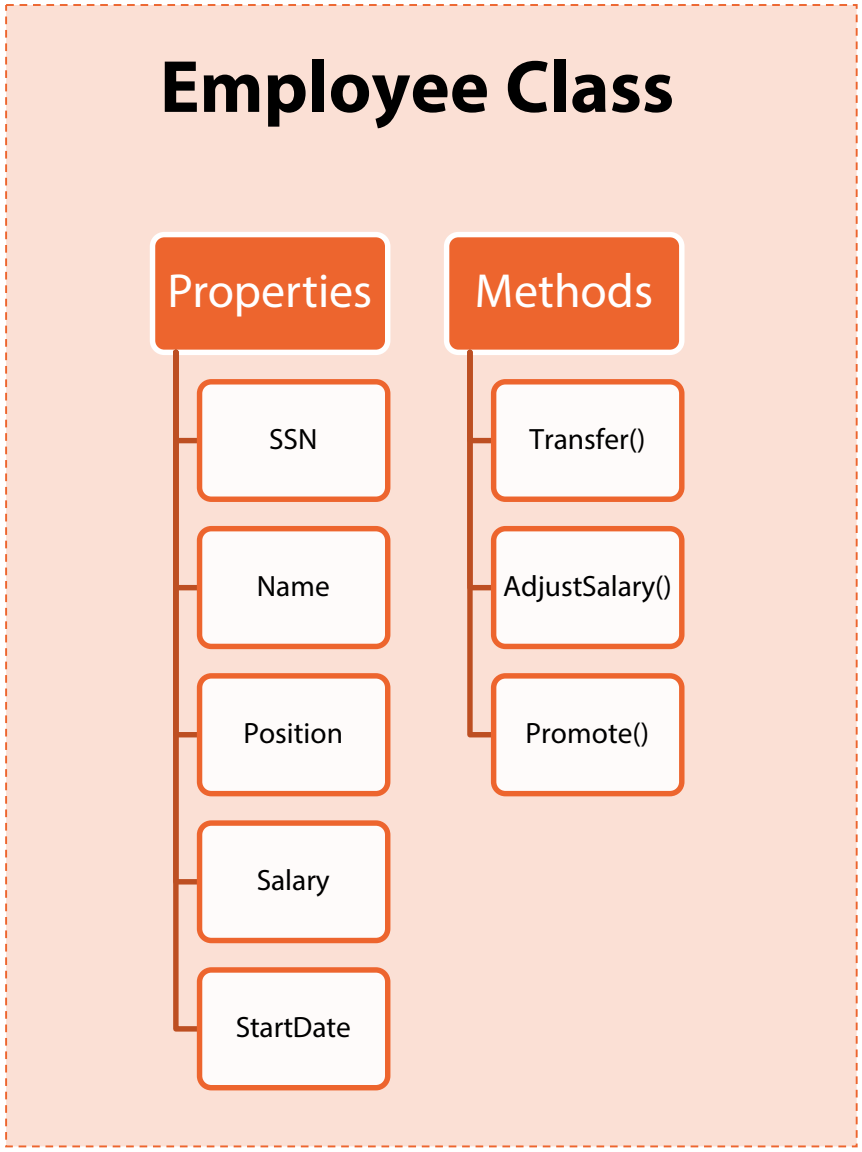


Leads to Chaos!

- **Nothing enforces relationships between items**
- **What if two or more procedures modify salary data?**
  - Changes to rules require updating program logic in several places
- **Encapsulating rules makes management easier**
- **All references to data must be associated with a given object**
  - Always know what “thing” you’re operating on
- **Processes that affect an object are defined as part of the object**

Working with Objects

- **Consumers of the object are insulated from inner workings**
  - Cannot modify properties unless you let them
- **Code modifications occur only in one place**
- **A class defines the methods, properties, and events associated with a type of "thing"**
- **Application interacts with instances of the class**
  - Must obtain a reference to the object in order to interact


Employee Class
Properties
SSN
Name
Position
Salary
StartDate
Methods
Transfer()
AdjustSalary()
Promote()

Try Out a Built-In Class

- Word's Application class
- Word automatically creates instance of the class for you, and only allows one instance
- Provides various properties and methods

DEMO 1

- Interact with Word's Application object

Comparing Classes and Objects

- **Class modules are like document templates**
  - Define the characteristics and behavior of an object
  - Can't use them to manipulate the characteristics or run code
  - Like Word document templates, or PowerPoint presentation templates
- **Class instances are like documents**
  - To use a class, must create an instance of the class
  - Like creating a document based on a template
  - Each instance contains the properties and methods of the class
    - You can manipulate and interact with these instances in code
    - Just like creating a Word document from a template
  - Each instance maintains its own individual set of property values

Another Analogy

Mammal Class

Type Property
Speak() method

Dog

Sheep

Cat

Using a Class

- **In order to use class, must create instance**
- **Just as you can't live in a blueprint for a house**
  - Or drive the design for a car
  - Must create instance of those plans in order to use
  - Must create instance of class to work with the object
- **Can't call methods or interact with properties of a class**
  - Until you create an instance

Creating an Instance

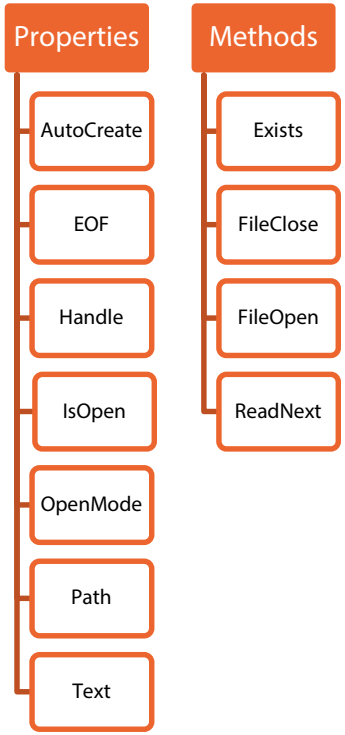
- **Declare an object variable based on the class**
- **Variable stores references to class instance**
- **Object variables follow same rules as normal variables**
  - Can use Dim, Private, Public, or Global
- **Then, use Set statement with New keyword:**

```
Dim myTextFile As TextFile
Set myTextFile = New TextFile
```

- **New keyword creates instance of class; Set keyword assigns**

Creating a Useful Class

- **To demonstrate basic techniques…**
  - Next example creates a TextFile class
  - Example class is read-only
  - Could easily extend to allow writing to files
- **Discussion will continue to add features**



DEMO 2

- Create TextFile class module
- Name class
- Examine events

Creating a Property

- **Two ways to create a property**
- **Easiest: Create a public property in the class module**
  - Limits usefulness—can't run any code to set or retrieve value
- **More complex: Create property procedure (later)**
  - Allows you to run code in reaction to getting or setting property
- **AutoCreate property controls whether a new file is automatically created if it doesn't already exist**

```
Public AutoCreate As Boolean
```

Why Not Use Public Variables?

- **Several drawbacks:**
  - Class has no way of knowing if an outside process has changed value
    - What if you need to take action in response to change?
  - Can't restrict property or perform data validation
    - Maybe want to restrict a person's age to numbers between 0 and 110
  - Can't create read-only (or write-only) properties
- **Solution?**
  - Create a pair of property procedures, a topic for later
- **You can use Private variables**
  - But only available to code within the class

DEMO 3

- **Add AutoCreate property**

Creating a Method

- **Creating a public variable creates a property**
- **Create a public sub or function creates a method**
- **TextFile class provides FileOpen, FileClose, Exists, and ReadNext methods**
- **Would have been nice to call methods Open and Close**
  - Reserved words in VBA, can't use those!
- **Could discuss whether Exists should be method or read-only property**
  - Determines if the file actually exists

Referring to Class Members Using Me

- **Sometimes, need to refer to class properties or methods from within class**
- **Can refer to internal property variables, or call procedures directly**
- **Might want to consider using Me prefix (Me.PropertyName)**
  - "Me" refers to current instance of the class
  - Allows code to use same object-oriented constructs that outside code would use
  - Generally, a good idea

DEMO 4

- Create methods

Introducing Property Procedures

- **Created property using public variable**
  - Consumers access using object.property syntax
- **No way class can know when property is set or retrieved**
  - To solve, use property procedures
- **Three varieties of property procedures:**
  - Property Get – get value of scalar or object property
  - Property Let – set value of scalar property
  - Property Set – set reference for object property

Why Use Property Procedures?

- **Property Get procedure allows you to supply a property value**
  - Makes it easy to take action when retrieving property value
  - Perhaps calculate a new value for the property
- **Property Let procedure allows you to accept a new property value**
  - Make it easy to run code in reaction to setting property
  - Perhaps handle error conditions
- **Property Set procedure allows you to set new property reference**
  - Caller passes in an object, code assigns it to internal reference

Property Get

- Simple procedure returning value of property
- Often simply returns value of private internal variable

```
Property Get TheProperty() As Integer
  TheProperty = someValue
End Property

' To retrieve:
Dim obj As ObjectType
Dim value As Integer

Set obj = New ObjectType
value = obj.TheProperty
```

Property Let

- Assignment of scalar property value
- Value on right-hand side of = passed as parameter

```
Property Let TheProperty(value As Integer)
  someValue = value
End Property

' To set:
Dim obj As ObjectType
Set obj = New ObjectType
obj.TheProperty = 12
```

DEMO 5

- Create property procedures

Property Set

- **Property Set procedures allow you to set a property as a reference to another object**
  - Property of one class that is a reference to another object
  - Must use Property Set to assign the reference
- **Property Let and Property Set are not interchangeable in VBA**
- **Use Property Get to retrieve reference**
  - No separate “getter” for object properties

Using Property Set

```
Property Set TheObjectProperty(value As Object)
  Set someValue = value
End Property

' To set:
Dim obj As ObjectType
Dim objValue As SecondObjectType

Set obj = New ObjectType
Set objValue = New SecondObjectType

Set obj.TheObjectProperty = objValue
```

DEMO 6

- Add Property Set for SaveFile property

Read-Only/Write-Only Properties

- **Write-only property rarely used, but possible**
  - Simply provide only a Property Let/Set procedure
- **Read-only properties more useful**
  - Provide only a Property Get procedure
  - Retrieve internal value, or calculate
- **TextFile Handle property is read-only**
  - No way for external code to set file handle value

Passing Parameters to Property Get

- **Can pass parameters to Property Let procedure**

```
' Property in Payroll class
Property Get PayDay(week As Integer) As Date
  PayDay = ProcedureToCalculatePayDay(week)
End Property
```

- **To get value of PayDay property in Payroll class:**

```
Dim pr As Payroll
Set pr = New Payroll

Dim thePayDay As Date
thePayDay = pr.PayDay(12)
```

Passing Parameters to Property Let

- **As with Property Get, can pass parameters**
- **A bit more complicated**
  - Normally pass parameter to Property Let using =
    - Object.TheProperty = value
    - Property Let TheProperty(value)
  - When setting property, value after equals sign passed in as parameter
  - If multiple parameters
    - Value after equals sign becomes final parameter
    - Subsequent parameters passed as parameters to property procedure
- **Code is similar for Property Set**

Passing Parameters to Property Let

```
' Property in Payroll class
Property Let PayDay(week As Integer, _
    year as Integer, newPayDay As Date)

    ' Set payroll date given the year and the week.
    ' This code is up to you…
End Property

' To set the value:
Dim pr As Payroll
Set pr = New Payroll

Pr.PayDay(12, 2015) = #3/22/2015#
```

Passing Parameters to Property Procedures

- **Possible, generally not done**
- **Instead, can create methods with names like GetProperty and SetProperty**
  - GetPayDay and SetPayDay, in sample case

Creating Enumerated Values

- **In example, OpenMode accepts integer**
- **Would be easier to code if accepted an enumerated value**
  - Could select from possible values in IntelliSense
  - Makes code easier to read
- **Simple to create**
- **Rather than list of integer constants representing open mode**
  - Create Enum with a name
  - Elements of Enum numbered starting at 0
  - Sequentially increment
  - Can override value of any item; subsequent values increment
- **Enumerated values limited to long integers!**

Creating Enumerated Values

- **Simple enumeration:**

```
Public Enum TextFileOpenMode
  OpenReadOnly
  OpenReadWrite
  OpenAppend
End Enum
```

- **Can specify values:**

```
Public Enum TextFileOpenMode
  OpenReadOnly = 3
  OpenReadWrite = 5
  OpenAppend
End Enum
```

DEMO 7

- Create enumerated value

Initialize and Terminate Events

- **Unlike normal modules**
  - Class modules provide Initialize and Terminate events
  - Events allow you to control what happens as instance is created and destroyed
- **Initialize event occurs when instance is first created**
  - Allows you to initialize property values or create references to other objects
- **Terminate event occurs when last reference to the object is released**
  - Either when variables go out of scope, or when explicitly set to Nothing
  - Allows you to perform cleanup tasks

DEMO 8

- Add code to Initialize and Terminate event handlers

Set Reference to Nothing?

- **VBA handles reference counting**
  - When last reference to an object is released, object removed from memory
  - Set variable to Nothing to release its reference
- **When variable goes out of scope, VBA sets its reference to Nothing**
- **Do you need to set the reference to Nothing?**
- **Only if you want to explicitly decrement the reference count**
  - If variable goes out of scope, no point
  - Can't hurt

Condense Instantiation to One Line?

- **Normally, create and instantiate object reference in two steps:**

```
Dim tf as TextFile
Set tf = New TextFile
```

- **Lets you control when object gets instantiated**
- **Can also accomplish this in one line, but not quite the same:**

```
Dim tf as New TextFile
```

- **May seem more efficient, but generally not a good idea in VBA**
- **Allows VBA runtime to create instance when object is first used**
  - Not clear from code when that instantiation occurs
  - If code in Initialize event, unclear when that code will run

Summary

- **Simple to create your own classes**
- **Behave and interact just like built-in classes**
- **Allows you to encapsulate behavior**
  - Allows you to “hide” code from callers
- **Note that VBA not even close to fully object-oriented**
  - No support for inheritance, for example
  - But classes can still make it far easier to create and manage code